②

NAVAL POSTGRADUATE SCHOOL

Monterey, California

THESIS

ACCESSING NETWORK DATABASES VIA SQL TRANSACTIONS IN A MULTI-MODEL DATABASE SYSTEM

by

Dennis A. Walpole

and

Alphonso L. Woods

December 1989

Thesis Advisor: David K. Hsiao

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Naval Postgraduate School 6b. OFFICE SYMBOL *(If applicable)* 37	7a. NAME OF MONITORING ORGANIZATION Naval Postgraduate School
6c. ADDRESS *(City, State, and ZIP Code)* Monterey, California 93943-5000	7b. ADDRESS *(City, State, and ZIP Code)* Monterey, California 93943-5000
8a. NAME OF FUNDING/SPONSORING ORGANIZATION 8b. OFFICE SYMBOL *(If applicable)*	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS *(City, State, and ZIP Code)*	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. | PROJECT NO. | TASK NO. | WORK UNIT ACCESSION NO.

11. TITLE *(Include Security Classification)*
ACCESSING NETWORK DATABASES VIA SQL TRANSACTIONS IN A MULTI-MODEL DATABASE SYSTEM

12. PERSONAL AUTHOR(S)
Walpole, Dennis A. and Woods, Alphonso L.

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT *(Year, Month, Day)*	15. PAGE COUNT
Master's Thesis	FROM ______ TO ______	1989, December	77

16. SUPPLEMENTARY NOTATION
The views expressed in this thesis are those of the authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

17. COSATI CODES			18. SUBJECT TERMS *(Continue on reverse if necessary and identify by block number)*
FIELD	GROUP	SUB-GROUP	Multi-Backend Database System (MBDS) Multi-Lingual Database System (MLDS)

19. ABSTRACT *(Continue on reverse if necessary and identify by block number)*

Traditional approaches to database-system design and implementation involve single-model, single-language database systems with their inherent lack of flexibility and extensibility. An alternative to the traditional approach to database-system design and implementation is the multi-lingual database system (MLDS). This approach allows the user with the user's familiar data language to access and update one or more unfamiliar databases in different data models as if they are in the user's familiar data model. Thus, MLDS has the flexibility and extensibility in database accesses.

In this thesis, we present a methodology for the relational user to access and update network databases with a relational data language. Specifically, we designed an interface for allowing the relations/SQL user to access a network database via SQL transactions. This thesis further extends the functionality of MLDS. (KR) ←

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT [✓ UNCLASSIFIED/UNLIMITED] ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL Prof. David K. Hsiao	22b. TELEPHONE *(Include Area Code)* (408) 646-2253 22c. OFFICE SYMBOL Code 52Hq

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted. All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE

☆ U.S. Government Printing Office: 1986—606-24

UNCLASSIFIED

Accessing Network Databases via SQL Transactions in a Multi-Model Database System

by

Dennis A. Walpole
Lieutenant Commander, United States Navy
B.A., University of New Mexico, 1977

and

Alphonso L. Woods
Lieutenant, United States Navy
B.S., Prairie View A & M University, 1984

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
December 1989

Authors: [illegible] [illegible]
Dennis A. Walpole Alphonso L. Woods

Approved by: [illegible]
David K. Hsiao, Thesis Advisor

[illegible]
C. Thomas Wu, Second Reader

[illegible]
Robert B. McGhee, Chairman
Department of Computer Science

ABSTRACT

Traditional approaches to database-system design and implementation involve single-model, single-language database systems with their inherent lack of flexibility and extensibility. An alternative to the traditional approach to database-system design and implementation is the multilingual database system (MLDS). This approach allows the user with the user's familiar data language to access and update one or more unfamiliar databases in different data models as if they are in the user's familiar data model. Thus, MLDS has the flexibility and extensibility in database accesses.

In this thesis, we present a methodology for the relational user to access and update network databases with a relational data language. Specifically, we designed an interface for allowing the recreational/SQL user to access a network database via SQL transactions. This thesis further extends the functionality of MLDS.

TABLE OF CONTENTS

ACKNOWLEDGMENTS

We would like to thank our thesis advisor, Professor David Hsiao, for all the long hours of lectures on MBDS. Your expert knowledge of MBDS gave us the needed foundation to make our thesis a reality. We also would like to thank him for introducing us to the MBDS concept. This concept has made our research an enjoyable and worthwhile experience.

We would like to thank Thomas Chu for all the technical help given to us in debugging the system code. Your help was very instrumental in our thesis success.

I. INTRODUCTION

A. AN OVERVIEW

Databases have been an integral part of our society ever since records have been kept. Probably the most visible example is the merchants of old. They kept records of their customer's transactions and the balance of payments. The tax collector must have had a database to know who were in the district and what they had paid in the past. As times moved on databases were still kept by hand and the only access to them was with the user of the databases. Of course, copies could be made by hand; however, the point then was that the user would only make copies as a back-up for himself, since it was a time-consuming process.

As the computer was being developed, the databases were no longer updated by hand. Although the computer was initially thought of as a means to perform mathematical calculations that were too complex to be done by hand efficiently, the computer was eventually thought of as a means to store and retrieve data.

Database systems were therefore designed. The concept of massive storage capabilities and on-line manipulation of databases have brought on a myriad of database systems. Each database-system design was influenced by the type of databases utilized. Therefore, in a large organization the result was that it acquired more than one type of database due to diverse applications. Each database type had its unique language that must be used to access that type of databases. This situation created a problem for the organization. To access databases of the organization a user must know all the respective types and languages associated with the databases.

As the society has progressed in improving database usage the computer has provided for direct manipulation via complex queries which can be done in seconds that would take hours or days to do by hand. Further, the storage space required has vastly reduced and most importantly users may now have on-line data to work with. The problem is that these database systems and their databases are heterogeneous in that if a user wants information in a database system whose data model and data language are not familiar to the user the user cannot access and query the database.

An approach conceived by Demurjian and Hsiao [Ref. 1] solves the user access problem and reduces the maintenance costs of database systems. The concept is to create a database system that supports more than one database model and then allow users to access any database in the system and manipulate the database in the user's own familiar language. The first stage in the process is to allow a user to access multiple databases if the user is familiar with the languages required to use these databases. This system is the multi-lingual database system (MLDS) [Ref. 2]. Thus, it is no longer necessary to have a large number of heterogeneous database systems to support heterogeneous database applications. The benefit in cost is that hardware is only required to support a single system, i.e., MLDS. Further, maintaining a single system is more efficient than training and providing maintenance for many heterogeneous systems. The second stage in the process is to provide cross-model accessing. More specifically, MLDS allows for direct accessing of multiple databases because all the data respond to the same data language. The user's data language is translated into the attribute-based data language (ABDL) and the database created in the user's data model is stored in the attribute-based data model (ABDM). This capability allows the user to access a database whose data model is foreign to the user with the user's own familiar data language. Thus, the cross-model accessing capability will extend MLDS into a multi-model database system (MMDS) [Ref. 3].

B. THE MULTI-LINGUAL DATABASE SYSTEM (MLDS)

The multi-lingual database system has been described in many theses prior to this one and we also briefly trace the path that a user will take when accessing the system. The structure of the multi-lingual database system is depicted in Figure 1. In accessing a database the user uses a user data language (UDL) that corresponds with the user data

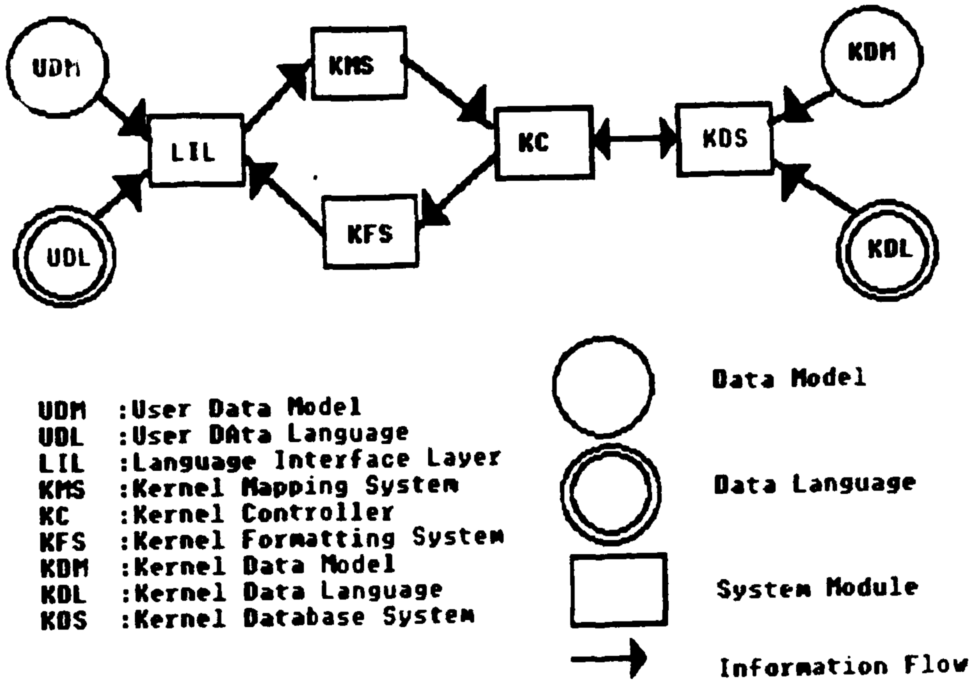


Figure 1. The Multi-Lingual Database System

model (UDM). The language-interface layer (LIL) identifies which of the two possible transaction types is requested and sends it on to the kernel mapping system (KMS). If the user wants to create a new database, KMS modifies UDM to the kernel data model (KDM); if the user wants to use an existing database, KMS modifies UDL to a kernel data language (KDL). The transformed request is then routed to the kernel database system (KDS) via the kernel controller (KC). When KDS has completed the task desired by the user, it then sends the result back to LIL in a format that UDM recognizes. The kernel formatting system (KFS) receives the result from KDS via the KC, reverses the process performed by KMS, and sends the result to LIL where the user receives the result.

LIL is unique to a pair of UDM and UDL. In a multi-lingual database system, a separate LIL is required for each pair of UDM and UDL; however, KDS is shared by all UDM. Our system currently has language interfaces for the relational model and SQL language, the hierarchical model and DL/I language, and the network model and COADYSYL-DML language. KDS is used to access the actual database and to manipulate the database by one of the language interfaces. All databases are stored in the form of the attribute-based data model (ABDM) and accessed by the attribute-based data language (ABDL) denoted by KDM and KDL, respectively, Figure 2 shows how UDMs and UDLs as separate entities access the same database structure. The present system started as a concept by Hsiao [Refs. 4,5] and reviewed by Rollins [Ref. 6]. Students at the Naval Postgraduate School have used papers and theses that provide guidance on the mappings of relational [Ref. 7], hierarchical [Ref. 8], network [Ref. 9] and functional [Ref. 10] databases to the attribute-based model. Students have mapped and implemented the following language interfaces: SQL-to-ABDL [Refs. 11,12], DL/I-to-ABDL [Refs. 13,14], CODASYL-DML-to-ABDL [Refs. 15,16]. The implementations have been modified by professionals and are

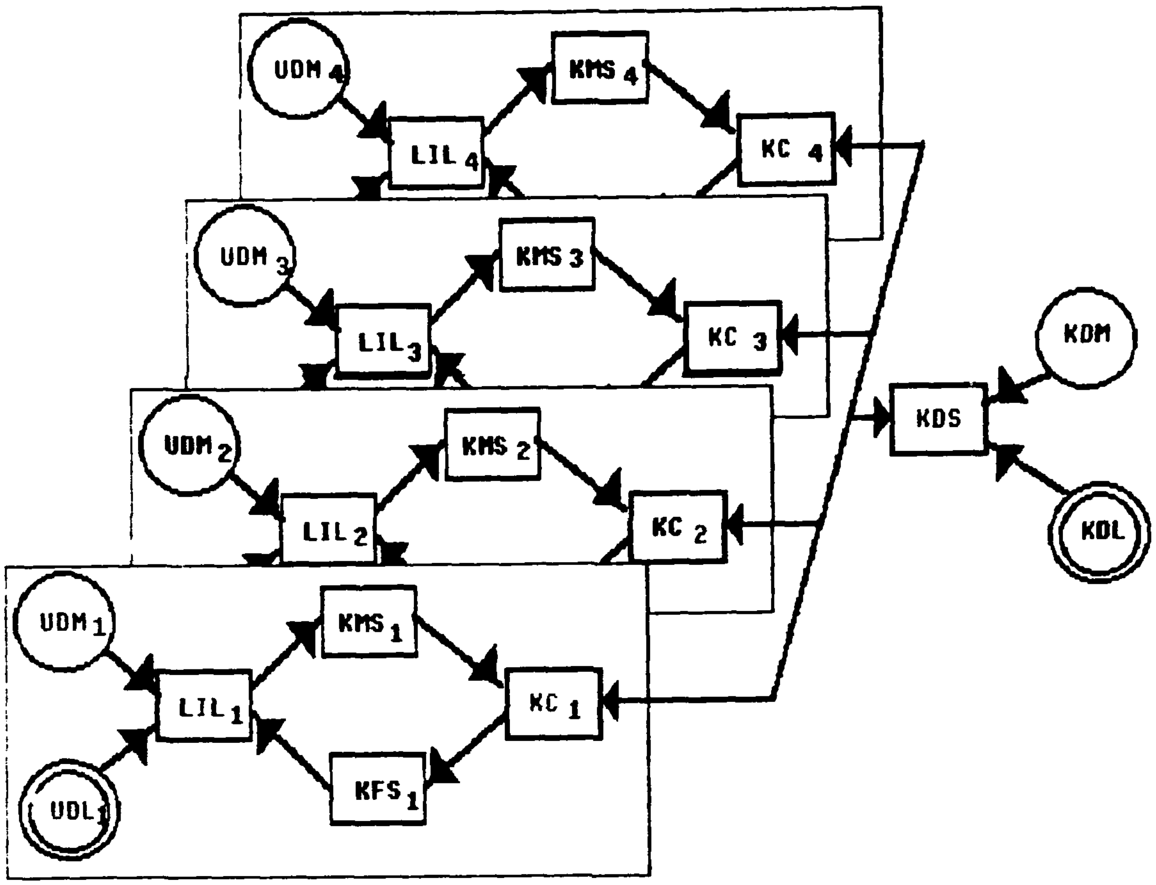


Figure 2. Multiple Language Interface

operational. Also, the language interface DAPLEX-to-ABDL has been designed [Ref. 17] but not implemented.

C. THE MULTI-MODEL DATABASE SYSTEM (MMDS)

The result of Zawis' thesis [Ref. 18] transformed MLDS into MMDS [Ref. 3]. His thesis provided a cross-model accessing capability to any relational user who wants to access a hierarchical database. Figure 3 shows the MMDS structure. The relational/SQL interface was modified by Zawis to include a hierarchical database type in the

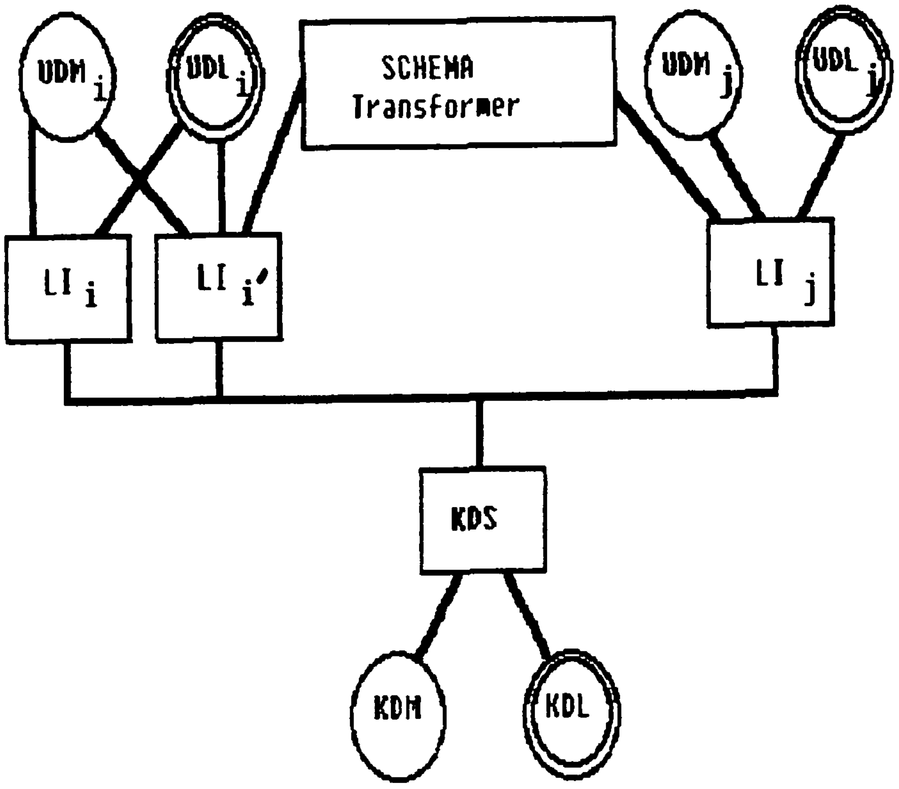


Figure 3. The Multi-Model Database

relational schema. It provides for identification of the data model of a database being accessed. This is needed since the original relational/SQL interface is based on only one database model, i.e., relational. When a database is accessed the new interface will search for the schema of the database. When found the mapping schema will identify the database model. On the basis of the database model the interface branches to the appropriate procedures that allow manipulation of the database. This solution to cross-model accessing capability is effective in the current environment and will be expanded in future work. This process is a major step for mono-language users in a multi-language database environment. The multi-model database system is thoroughly discussed in [Ref. 18].

D. THE MULTI-BACKEND DATABASE SYSTEM (MBDS)

The multi-backend database system (MBDS) is not a factor in our research. However, it is a part of the system used to support our interface. We will briefly discuss the system to provide the reader with an overview of the total system. There exists more detailed discussion of MBDS in [Ref. 19] and [Ref. 20]. MBDS has provided a solution to a problem that exists in the conventional approach to database performance. As in Figure 4, each backend has, in addition to its own disk system, its own processor which are linked by a communications bus and controlled by a controller. This set up provides for parallelism which increases its performance gains and capacity growth. The gains and growth can be increased by the addition of more backends.

E. THE THESIS ORGANIZATION

We are adding the capability of using the relational language, SQL, to access and manipulate a network database. Prior to the thesis work, network databases are only accessible through the network language, i.e., CODASYL-DML. Our task is to design and implement the cross-model accessing capability between a relational language and

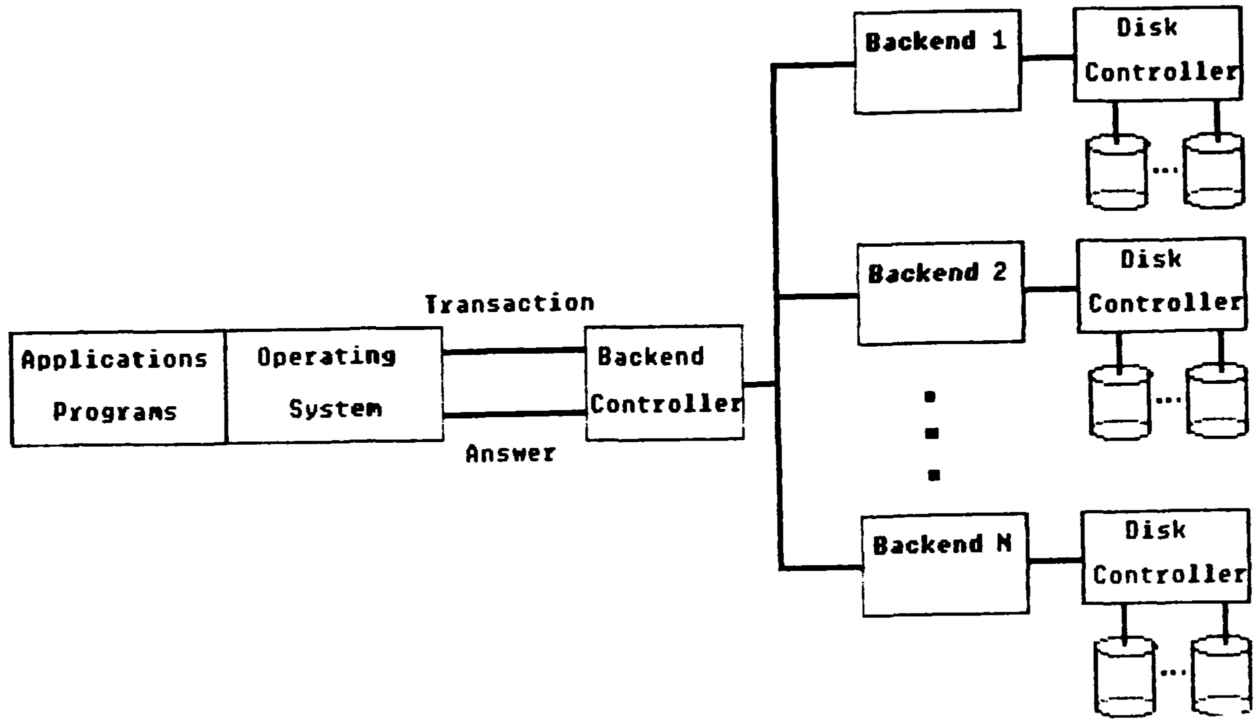


Figure 4. The Multi-Backend Database System

network databases. We do not disrupt the existing cross-model accessing capability as implemented by Zawis. In Chapter II we discuss the data models with which we are concerned. Those are the attribute-based data model, the relation data model and the network data model. In Chapter III we discuss the strategies considered by Rodek and Zawis in implementing cross-model accessing capabilities. In Chapter IV we discuss our design and implementation on mapping a network database schema into a relational database schema. In Chapter V, we discuss our design and implementation of the network/SQL interface and modifications needed to support transaction and database integrity. In Chapter VI we give our conclusions and remarks.

II. THE DATA MODELS

In this chapter, we discuss the three data models and their corresponding data languages which are used in our research. Since the database being accessed is stored as an attribute-based database, Section A provides an overview of the attribute-based data model. Section B looks at the user's model which, in this case, is the relational data model. The final model covered in our research is the network data model. The distinction is that in our system we do not access a network database stored as a network database, but a network database that was transformed into an equivalent attribute-based database for storage. The CODASYL data manipulation language is not covered, since it is not relevant to our research.

A. THE ATTRIBUTE-BASED DATA MODEL (ABDM) AND LANGUAGE (ABDL)

The attribute-based data model (ABDM) originated in [Ref. 4]. ABDM was implemented into the kernal database system (KDS) discussed in Chapter I. The attribute-based data model has two types of data. They are the base data and the meta data; together they form the database.

1. The Base Data

A database is a collection of files. Every file has records. A record is a collection of attribute-value pairs (keywords) and the record body. The attribute-value pair is a member of a Cartesian product of the attribute set and the value domain of the attribute. Each attribute-value pair can only exist once in a record. No two attribute-value pairs in a record may have the same attribute. Directory keywords of a record are attribute-value pairs or attribute-value ranges that are stored in a directory. The attribute-value pairs in the records which are not kept in a directory are called non-directory keywords. The remainder of a record is textual information and comprises the record body. The following is an example of a record in the attribute-based data model.

```
  (<FILE,Suppliers>,<SNO,S1>,<SNAME,Jones>,<CITY,Monterey>,
         {Parts Supplied})
```

The parentheses enclose a record. The angle brackets enclose the attribute-valued pairs and the squiggly brackets constitute the record body.

2. The Meta Data

The meta data are stored information about the base data. More specifically, the directory is the collection of the meta data for a database. In the directory there are attributes, descriptors and clusters. Attributes are as presented previously. A descriptor describes the range of values or an exact value of an attribute. A cluster is a group of records that satisfy a unique set of descriptors. We use tables to maintain the directory. These are the attribute table (AT), the descriptor-to-descriptor-id- table (DDIT) and the cluster definition table (CDT). These tables make up the directory, an example of which is shown in Figure 5. The attribute types are A, B and C. Type A are those with variable value- ranges. Type B are those with unique values. Type C have unique values as type B; however, the attribute value is entered at the record input times. Type-C attribute values are added to DDIT if they do not already exist; however, type A and B attribute values are fixed at the database creation time and will not change as new records are being inserted.

3. <u>The Attribute-Based Data Language (ABDL)</u>

A brief description of the five primary operations of ABDL is as follows:

To insert a new record in a database, INSERT must proceed the record to be inserted, i.e., INSERT (record). An example of an insert is:

```
INSERT(<FILE=Supplier>,<SNO=S1>,<SNAME=Woods>,
<CITY=Monterey>)
```

A deletion can affect more than one record. To specify the set of records to be deleted, we use a query, i.e., DELETE (query). Therefore, a delete is different from an insert. The former takes a query; the latter includes a record.

As an example we can delete all suppliers from Monterey.

Attribute	Attribute Type	DDIT Entry
POPULATION	A	D11
CITY	C	D21
FILE	B	D31

(a) An Attribute Table (AT).

Id	Descriptor
D11	$0 \leqslant$ POPULATION $\leqslant 50000$
D12	$50001 \leqslant$ POPULATION $\leqslant 100000$
D13	$100001 \leqslant$ POPULATION $\leqslant 250000$
D14	$250001 \leqslant$ POPULATION $\leqslant 1000000$
D21	CITY = Cumberland
D22	CITY = Columbus
D23	CITY = Monterey
D24	CITY = Toronto
D31	FILE = CanadaCensus
D32	FILE = USCensus

Dij: Descriptor j for attribute i.

(b) A Descriptor-to-Descriptor-Id Table (DDIT).

Id	Desc-Id Set	Rec-Id.
C1	{D11,D21,D32}	R1,R2
C2	{D14,D22,D32}	R3
C3	{D12,D23,D32}	R4
C4	{D14,D24,D31}	R5

(c) A Cluster-Definition Table (CDT).

Figure 5. The Directory Tables

DELETE((FILE=Suppliers) and (CITY=Monterey))

UPDATE is used to modify a record or a set of records in a database. The request consists of two parts: a query and the modifier. The query indicates which part of the database is to be modified and the modifier specifies how the database is to be updated, i.e., UPDATE((Query) (Modifier)).

The following is an example of an UPDATE request:

UPDATE((FILE=Suppliers) and (SNAME=Jones)(CITY=Carmel))

This UPDATE request will update the Supplier named Jones as being located in Carmel.

RETRIEVE is used to gather information from a database. The request contains a query, a target-list and a by-clause. The query indicates which records are to be retrieved. The target-list is a list of attributes that are to be retrieved. The by-clause is optional and groups records, i.e., RETRIEVE((Query)(Target-list)) (by-clause).

The following is an example of a RETRIEVE request:

RETRIEVE(FILE=Suppliers)(SNAME) BY SNO

This request will retrieve all the names of suppliers in the Supplier file in order of the SNO.

RETRIEVE COMMON is used to merge two files into one where records from the respective files have common attribute values. The format of the request is as follows:

```
RETRIEVE(Query-1)(Target-list-1)
COMMON(Attribute-1,Attribute-2)
RETRIEVE(Query-2)(Target-list-2)
```

Where the common attribute values are specified by their attributes, i.e., attribute-1 for the first RETRIEVE and attribute-2 for the second RETRIEVE.

An example of a RETRIEVE COMMON query is as follows:

```
RETRIEVE(FILE=Supplier)(CITY)
COMMON(CITY,CITY)
RETRIEVE(FILE=Part-location)(CITY)
```

This query retrieves all suppliers and parts that are located in the same city.

B. THE RELATIONAL MODEL AND LANGUAGE

1. A Model Description

The relational model was proposed by E.F. Codd in 1970 [Ref. 21]. The model is a collection of tables that form a "flat" database, unlike the hierarchical and network databases that use a tree structure and a network, respectively. "An Introduction to Data Base Systems" by C.J. Date [Ref. 22] is recommended for further reading to provide an overview of relational database concepts.

A relational database is a collection of tables or relations that are equivalent to files. Within each table there are tuples which are records of the table. A tuple consists of a group of attribute values and all tuples in the table have the same attributes. Therefore, specific attribute names are the column headings of the table and the individual tuple are the rows.

The tables (relations) are not connected by any structure. The tables are identified by their unique table names (relation names). The key attributes uniquely identify the tuples. Thus, within the table no two tuples may have identical values for the key attributes.

2. The Data Manipulation Language (SQL)

The relational data language supported by our system is SQL. This is a widely-used relational data language. A description of SQL is not provided, but examples of the four basic queries are given. A comprehensive discussion of SQL can be found in Date [Ref. 22].

SELECT is the command used to retrieve attribute values from the database. The query is structured so that a set of attributes is specified for a relation where the specific attribute values are to be selected. The relation name is in the FROM clause. The optional WHERE clause defines the attribute names to match in the relation. If a match exists the attribute value(s) in the SELECT clause are displayed.

```
SELECT        attribute(s)
FROM          relation
WHERE         query
```

An example is as follows:

```
SELECT          CITY
FROM            Supplier
WHERE           SNAME= Jones
```

This request selects the value(s) of the attribute CITY from the relation Supplier where the SNAME is Jones.

The select command can be more complex and a select command from more than one relation is also possible. Our system at this time provides for up to two relations.

The INSERT request inserts a new tuple into an existing table. In this situation the command inserts attributes with values into a relation. The attribute order and number of attributes must match the relation exactly.

```
INSERT INTO Relation (attribute-names)
<attribute-values>
```

An example of an INSERT follows:

```
INSERT INTO Supplier (SNO,SNAME,CITY):
<'S1', 'Jones', 'Monterey'>
```

This command inserts S1,Jones and Monterey into the Supplier relation.

The DELETE command will remove one or more a tuples from a relation in a database. In the delete command structure all tuples with occurrences in a relation where the query is matched, will be deleted.

```
DELETE          relation
WHERE           query
```

An example of a DELETE command follows:

```
DELETE        Supplier
WHERE         SNO='S1'
```

This DELETE command removes all tuples where SNO is equal to S1 in the Supplier relation.

An UPDATE command is used to modify attribute values in one or more tuples. The structure of the UPDATE command is that a relation is updated in the modifier attribute with the new value where the query is true.

```
UPDATE        relation
SET           modifier
WHERE         query
```

An example of an UPDATE command is:

```
UPDATE        Supplier
SET           CITY=Memphis
WHERE         SNO=S1
```

This UPDATE command replaces the CITY attribute value with Memphis in the Supplier relation where SNO is equal to S1.

C. THE NETWORK DATA MODEL AND LANGUAGE

The network (CODASYL) data model is based on the concept of directed graphs. Graphs consist of nodes and arcs. <u>Data Base Management Systems</u> by Cardenas [Ref. 23] has an excellent introduction to the network (CODASYL) schema and architecture. The original design and implementation of a network database employed the most restrictive options. The options involve insertion, retention and set-selection. The structure of a network database as it is realized in the attribute-based form is well described by Wortherely [Ref. 15]. Our description follows his and Rodeck's [Ref. 10] work to maintain consistency.

1. A Model Description

The network (CODASYL) databases are networks of record types and set types, where records and sets are the entities which describe the databases. A record type in a CODASYL database is defined as a collection of hierarchically-related data item names or field names. A record is any occurrence of a record type and has specific values assigned to the data items named in the schema declarations. This implies that a record type is simply a generic name for all of the records that are described by the same schema. Set types in a CODASYL database indicate relationships between record types. They consist of a single record type called the owner record type, and zero or more record types called the member record types. Thus, a set type expresses explicit associations between different record types in the database. This characteristic makes it possible for a designer to model a large variety of real-world database management problems involving diverse record types [Ref. 15]. The many-to-many relationship is limited in that an owner record of a set type cannot be a member of the same set type.

Set types have occurrences just as record types do. Each occurrence of a set type has one occurrence of the owner record type and zero or more occurrences of each its member record types. The same restriction applies here in that a record occurrence cannot be present in two different occurrences of the same set type. This restriction emphasizes the pairwise disjointness of the set occurrences of a given set type [Ref. 15].

2. The Data Manipulation Language (CODASYL)

The CODASYL language is used to create a network database. However, this thesis is concerned only with accessing a network (CODASYL) database and not in the language that creates it, but in the relational language SQL. We therefore do not include a discussion of the CODASYL manipulation language. However, the book The Codasyl Approach to Data Base Management by T. William Olle [Ref. 24] provides a very comprehensive look at the CODASYL manipulation language.

3. The AB(Network) Database

In our implementation, a network database is stored in the ABDM form. Thus, the database looks like an attribute-based database. What distinguishes a network database, say, from a relational database which is also stored in the ABDM form is the presence of a network schema for the database. Similiarly, there is a relational schema for a relational database. To characterize our approach to the support of network databases via ABDM storages and network schema, we refer to our network databases, the AB(network) databases.

III. THE CROSS-MODEL ACCESSING CAPABILITY

A. THREE APPROACHES TO THE CAPABILITIES

As more databases proliferate with their associated data manipulation languages, the need for a more flexible database system is needed. MLDS discussed earlier offers more flexibility than conventional database systems by giving the user the capabilities to access heterogeneous databases based on different data models with the user's familiar data manipulation language. The solution is to have: (1) the capability to access any database with a generic data manipulation language, (2) the translation of the user's data language into the kernel data language, (3) the presence of a database schema for the database which is based on the user's familiar data model. This process is the concept of the Multi-Model Database System (MMDS). The scope of this thesis is specifically concerned with giving the user the ability to access and query a network database via SQL (a relational data language). In this case the user's familiar data model is relational (not network) and the user's familiar data language is SQL (relational, not Codasyl-DML). Nevertheless, we provide the user with the capability to access a network database as if it is a relational one.

Rodeck [Ref. 10] described various strategies for implementing MMDS. A summary of the proposed strategies is presented in the rest of this chapter.

1. <u>The High-Level Preprocessing Method</u>

This method is called high-level preprocessing because the processing occurs "above" the local interface. This means a user inputs a database name for processing. The local interface is searched for the database name, if not found locally, the other LIs are searched. When found in another LI, the schema transformer transforms the found schema into an equivalent local database schema based on the local data model. When the user queries the database via the transformed schema with the local data language (say SQL), the second component of LI, language translator, translates the queries (say, Codasyl-DML) into the equivalent queries in the local data language for accessing the database. The third component of LI is the result formatter. It formats the results of a query into a form the user can recognize. For example, if a user wanted to access a network database via SQL transactions, the results would be returned to the user in a table form, instead of the network form. Figure 6 depicts this processing strategy.

2. <u>The Mixed-Processing Strategy</u>

This method of processing differs from the previous method in that there is no language-translation step. There

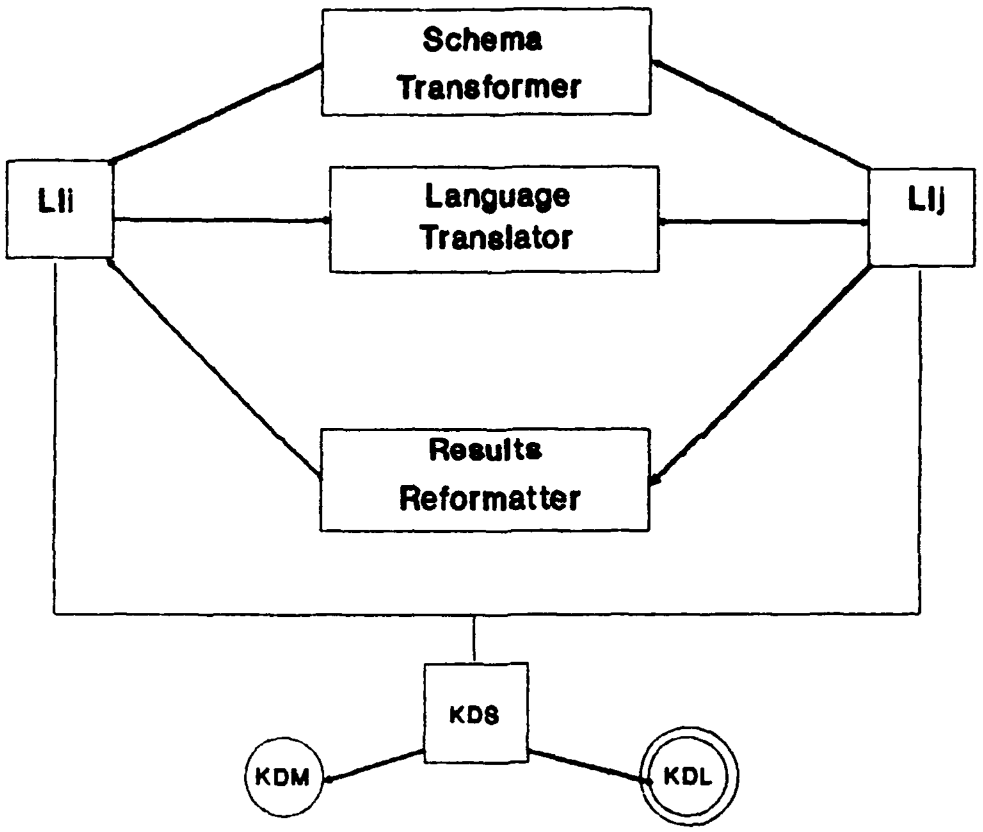


Figure 6. The High-Level Preprocessing Strategy

are two components involved in the process, the schema transformer and a second language interface. See Figure 7. Similar to the preprocessing strategy, when a user inputs a database name for processing, the local LI is searched; if the database is not found, other LIs are checked. After the database is found in another LI, a copy of its schema is transformed into an equivalent schema and placed in the local LI. When queries are entered against the transformed schema in the local data language, the local LI processes the request without the need of any translation. The

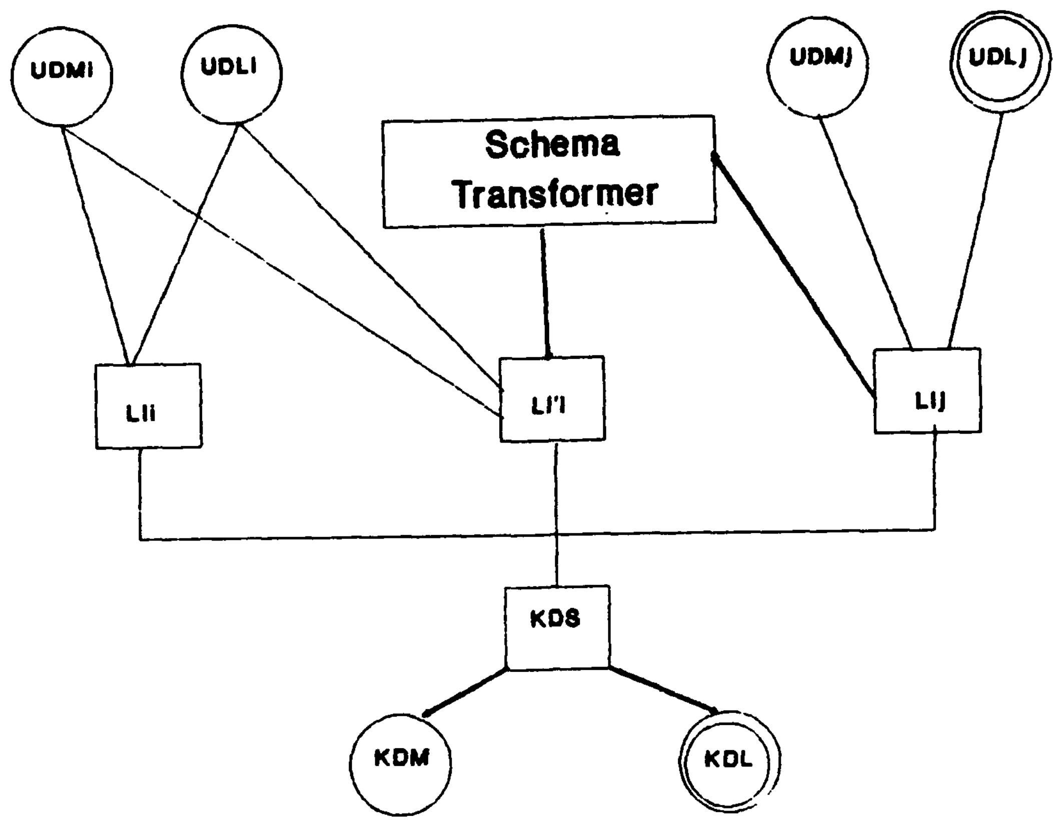


Figure 7. The Mixed-Processing Strategy

output also requires no reformatting, since it is in the form of the local database model.

3. The Postprocessing Strategy

The last strategy to be examined is the postprocessing strategy. In this strategy, the schema transformation takes place from the schema of the database to be accessed to the schema of the local LI, i.e., transforming a schema of a heterogeneous database into a schema in the form of the user's familiar data model. This strategy is called low-level because it occurs below the LI layer as illustrated in Figure 8. The language-translation portion of the strategy takes place in exactly the opposite direction to the schema transformation. The translation takes place from the local database language transactions to the equivalent database language transactions of the heterogeneous database. The result formatter outputs the results in the form of the local LI, i.e., use the transformed schema for output formats.

B. THE CHOSEN APPROACH

Because the postprocessing strategy involves both the schema transformation and transaction translation as discussed in Zawis [Ref. 18], it is too complicated to be used in our implementation.

The preprocessing methods involve translating the syntax of one data language into the syntax of another language; this is also a very complicated and time-consuming process. It is also ruled out as a viable strategy.

Thus, the mixed processing is chosen, since it does not require language translations for the same transaction. The last characteristic of the mixed-processing strategy is that this strategy requires less modification of existing code than the other strategies. The existing mixed-processing strategy for the network-to-relational transformation is very similar to the hierarchical-to-relational transformation. The major difference is the manner in which records are searched. In the hierarchal model, the user is

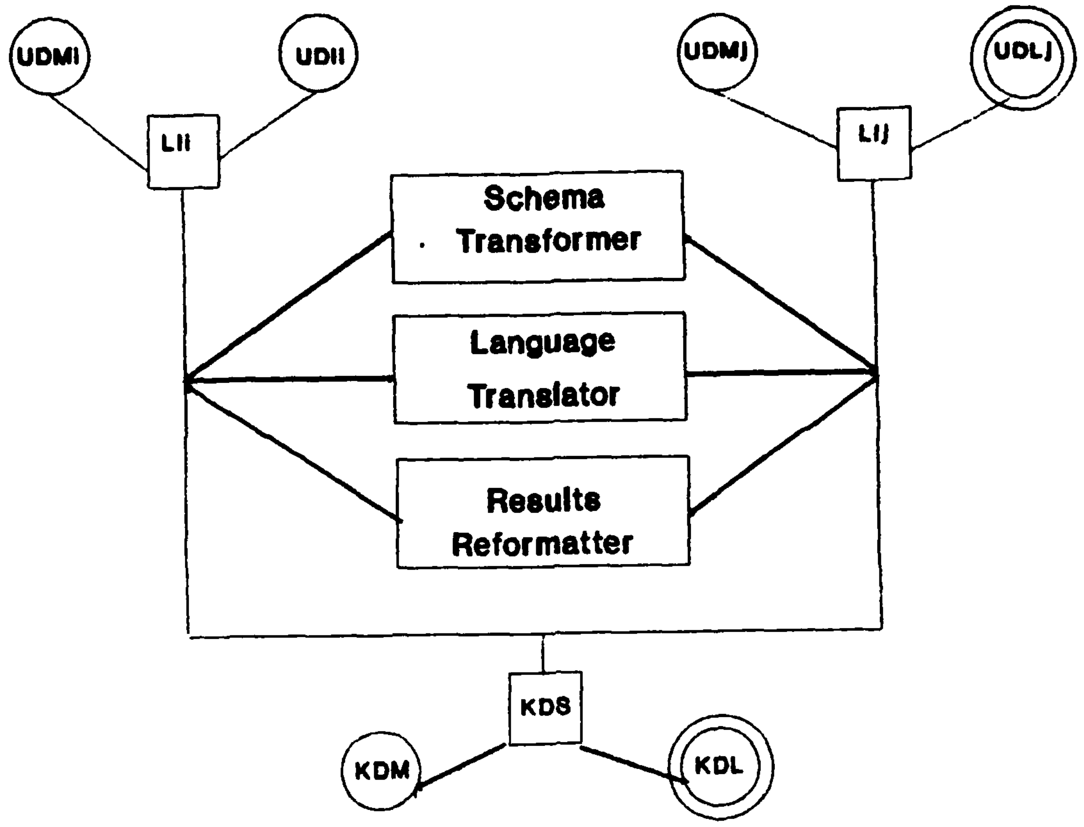


Figure 8. The Postprocessing Strategy

concerned with a single parent-child relationship. In the network model, the user has to search via sets and must also be concerned with records having multiple parents. However, once the desired record is found, the manner of retrievals is very similar. We can refer to Zawis' [Ref. 18] work on the hierarchical-to-relational for our work on the network-to-relational transformation.

IV. ON TRANSFORMING A NETWORK SCHEMA TO ITS RELATIONAL EQUIVALENT

A. THE DESIGN

The first step in the mixed-processing strategy is to perform the schema transformation. In our case, it is from a network schema to an equivalent relational schema. This is accomplished by translating the data relations in the network database to their equivalents in the relational model.

The network-to-relational transformation process will be illustrated by first describing a typical network database. Figure 9 illustrates the sample database to be transformed. The schema that describes the network database is "SPS." Figure 10 depicts this database definition. There are three record types called SA (Supplier), PA (Parts), SP (Supply). These record types will be functionally represented in a relational schema by tables. The "duplicates-are-not-allowed" declarations for SNO and PNO in respective SA and PA record types implies these attributes are key fields that uniquely describe an entity or record in question. Therefore, when desiring to insert, the program must check to see if the insert request has an attribute value that already exists in the database. The record attributes are defined by type and length.

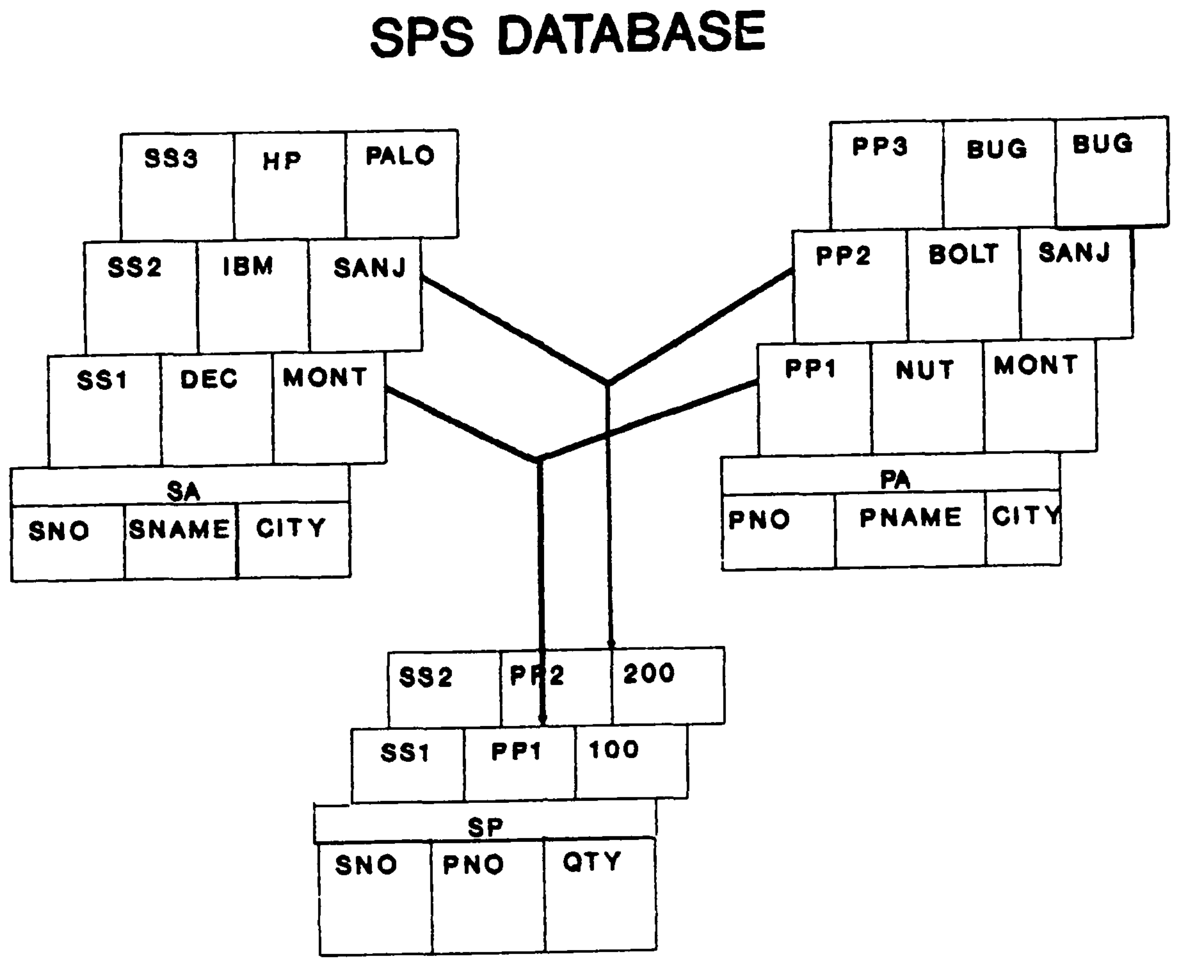


Figure 9. Supplier/Parts Network Database

The set types are now defined. Their purpose is to describe a relationship among record types. The two set types defined are SSP and PSP. Each set-type declaration will include the following: owner-record-type name, member-record-type name and insertion and retention rules. The particular details of each set type will differ, depending

```
schema name is SPS;
record name is SA;
  duplicates are not allowed for SNO;
    SNO ;  character 10.
    SNAME  ;  character 10.
record name is PA;
  duplicates are not allowed for PNO;
    PNO ;  character 10;
    CITY ;  character 10;
record name is SP;
    QTY ; fixed 4.
set name is SSP;
  owner is SA;
member is sp;
  insertion is automatic
  retention is fixed;
  set selection is by value of SNO in SA;
set name is psp;
  owner is PA;
  member is SP;
    insertion is automatic
    retention is fixed;
    set selection is by value of PNO in PA;
```

Figure 10. The Network Database Definition of a Sample Database

on the circumstances. The owner-name and member-name statements simply define a static relationship among existing records (i.e., occurrences) of the two record types.

The statement, insertion-is-automatic in set types SSP and PSP, means every record added or modified which represents a record type or subtype, must belong to a particular set. The statement, retention-is-fixed, requires a member record reflecting that a record subtype always belongs to the same owner-record type.

The last statement, set-selection-is-by-value, declares that when a record is inserted into a set, the set must be the current set type of SNO in SA and likewise PNO in PA. In simpler terms, this means each supplier and part will be inserted in the sets based on the owner record types.

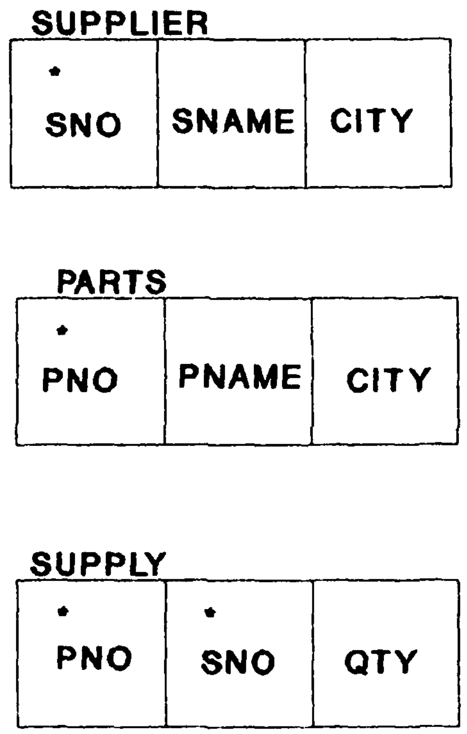
In transforming an existing network schema to a functional equivalent relational schema, various key issues must be observed. The relational database model has the characteristic commonly referred to as flatness. Flatness means that the tables (i.e., record types) have no (structural) relationships from each other. Whereas in a network or hierarchal database, the record and segment types are structurally linked. In the relational database environment, if structural relationships between tables are desired, data manipulation constructs such as the JOIN and VIEW are used.

As Zawis [Ref. 18] indicated in his work on the cross-model transformation (e.g., RELATIONAL-TO-HIERARCHICAL), there are key issues that also must be adhered to, in order to preserve the structural integrity of the mapped (i.e., in our case the network) schema to the equivalent (relational) schema.

One key issue is in the case of the network schema maintaining owner-set-member relationship in the network database. Similar to Zawis [Ref. 18] proposal for performing his transformation of hierarchal-to-relational, there are two methods that must be examined. His first method of schema transformation is to create a relational table for each relationship required in the given database. The fact that a network database has many-to-many relationships among its records; i.e., a record can have two owners and these owners can have parents, the proliferation of tables would make the representation not very cost effective. The second reason his method is not a sound method for our purposes is due to the numerous tree structures (hierarchies) in a network database, thus making queries against the database a very long and complicated process.

In our design, the method of schema transformation is to cascade data in key fields in the network records to form primary keys of equivalent relational tables. Figure 11 illustrates the cascading method. Key fields are defined as

A RELATIONAL SCHEMA



* indicates either the original or cascaded key field.

Figure 11. The Use of Cascading Keys to Preserve Owner-Member Relationships

fields that uniquely identify the corresponding records. They must remain consistent throughout the transformation. Figure 12 is an illustration of the transformed network database to its equivalent relational database.

```
database name = SPS
number of relations = 3, number of views = 0
database type = NETWORK

relation_name = SA, number of attributes = 3
    attr name = SNO      ,type = s, length = 10, key = TRUE
    attr name = SNAME    ,type = s, length = 10, key = FALSE
    attr name = CITY     ,type = s, length = 10, key = FALSE

relation_name = PA, number of attributes = 3
    attr name = PNO      ,type = s, length = 10,  key = TRUE
    attr name = SNO      ,type = s, length = 10, key = TRUE
    attr name = QTY      ,type = s length = 4, key = FALSE

relation_name = SP, number of attributes = 3
    attr name = PNO      ,type = s, length = 10, key = TRUE
    attr name = SNO      ,type = s, length = 10, key = TRUE
    attr name = QTY      ,type = i, length = 4 , key = FALSE
```

Figure 12. The SPS Database Mapped from a Network Schema to an Equivalent Relational Schema

B. AN IMPLEMENTATION

The implementation of the mixed-processing strategy required modification to the language interface layer (LIL), the kernel mapping system (KMS) and the kernel controller (KC). These modifications are made with little modifications of the existing Relational/SQL interface. This chapter will give a summary of the major data structures implemented in the new LIL as well as the flow of executions from LIL to the parser for the syntactical verification and execution of a SQL transaction.

1. The Language-Interface (LI) Structures

The language interface layer (LIL) is one of the most important layers in MMDS mainly because this layer directly links the user to the system. Upon an initial sign-on onto the system, there are numerous data structures that must be initiated in order to give the user access to the multiple databases. All the data structures in LIL will not be examined herein; however, the most important structures will be presented. The first structure present is the dbid_node in Figure 13. This structure points to a list of all the schemas that have been defined for all modeled databases. The structure also gives the user the ability to access all of the databases databases in the MMDS environment. After the user inputs a database name, the system searches the current list of databases based on the requested interface first. For example, if the user has

```
union dbid_node
  {
   struct  rel_dbid_node  *rel;
   struct  hie_dbid_node   *hie;
   struct  net_dbid_node   *net;
   struct  ent_dbid_node   *ent;
   }
```

Figure 13. The dbid_node Structure

requested a relational interface and then inputs a database name to be loaded onto the meta-data disk, the system would first search the list of relational database names via the dbid_node structure. If the database name is not found in the list of relational database names, it then searches the other defined database names.

The next data structure is the rel_dbid_node which points to the first relational database. This structure is depicted in Figure 14. It is the controlling data structure for all the schemas defined for the relational databases. The structure contains the name of the database, the number of relations, a pointer to the first relation, a pointer to the current relation, a pointer to the next database schema and, based on Zawis' first implementation of cross_modeling, a DBTYPE. The DBTYPE tells the user the name of the original database prior to the transformation.

The data structure rel_node shown in Figure 15 describes each of the relations in a database and is initialized with information available from the equivalent record data structures in the network schema. The relation

```
struct rel_dbid_node
   {
   char                              name[DBNLength + 1];
   int                               num_rel;
   struct    rel_node                *first_rel;
   struct    rel_node                *curr_rel;
   struct    rel_dbid_node           *next_db;
   int       dbtype
   }
```

Figure 14. The rel_dbid_node Structure

```
struct  rel_node
  {
  char                                          name[RNLength + 1];
  int                                           num_attr;
  struct          rattr_node                    *first_attr;
  struct          rattr_node                    *curr_attr;
  struct          rel_node                      *next_rel;
  }
```

Figure 15. The rel_node Structure

name is set equal to the network record name and pointers are set to the first attribute of the relation and to the next relation, if any, in the schema. The network records are mapped to this data structure via the set-member relationship.

The user_info data structure uniquely identifies a particular relational user. It also links the user to the linked list of other users on the multi-user environment. Figure 16 depicts this data structure.

The last data structure examined is the sql_info data structure. It contains pertinent information about the current database that the user is using, and information

```
struct user_info
   {
    char                          uid[UIDLength + 1];
    union     li_info             li_type;
    struct    user_info           *next_user;
    }
```

Figure 16. The user_info Structure

about the sql transaction. This structure is shown in Figure 17.

```
struct  sql_info
   {
   struct      curr_db_info          curr_db;
   struct      file_info             file;
   struct      ran_info              sql_tran;
   int                               operation;
   struct      ddl_info              *ddl_files;
   struct      tran_info             *abdl_tran;
   struct      kms_info              kms_data;
   union       kfs__info             kfs_data;
   union       kc_info               kc_data;
   int                               error;
   }
```

Figure 17. The sql_info Structure

2. <u>The Schema Transformation</u>

As previously mentioned, LIL is the most important layer in MBDS. It is from this layer the user logs onto the system and tells MBDS what type of tasks to accomplish. The user can load new databases, access previously created databases, and access information from an existing database. The flow of control is sequential. Control always returns to the menu-driven LIL. The user will always exit the system via the top level menu. The user can step back to top-level menu because an exit routine is provided at each level of menus.

The first menu, a user will see, gives the user options to load a new database, process an old database or return to the operating system. If he chooses to load new data into an existing database, the list of relational schemas is searched for the appropriate name. If the database name is found, the schema is loaded and processing may take place.

If the database name is not founded by searching the list of relational schemas, the system will search the list of schemas defined as network, hierarchal or functional. If found, it transforms the found schema to a functionally equivalent relational schema to facilitate SQL transactions. If the name of the selected database is located in the list of network databases, the data structure rel_dbid_node is appended to the end of the list of relational databases with the DBTYPE field having the value NET. The new data structure, rel_node, is now attached to the schema. The relational name of the SQL transaction is compared to the record name in a set type. If a match is attained, then the relation name in the SQL schema is equivalent to an owner-record name declared in a set type. If the relation is not found in the set type, then a comparison would be made on a member record. When found, the network record is set to equal to the SQL record and pointers are set to point the first attribute of the SQL record and the next relation.

The rattr_node data structure describes all the attributes associated with a relation. Figure 18 depicts this structure. Each attribute is represented by a unique rattr_node which contains a name, type, and length. The attributes are mapped directly from the network attribute node to the relational attribute node (struct rattr_node). If the network attribute is a key field, then the attribute is flagged in the relational schema with key attribute having a value of "true."

```
struct  rattr_node
  {
   char                                name[ANLength + 1];
   char                                type;
   int                                 length;
   int                                 key_flag;
   struct        rattr_node            *next;
   }
```

Figure 18. The rattr_node Structure

The cascading mechanism is used to map the key from the root nodes to the leaf nodes in the network structure. This is necessary in order to maintain the integrity of the network database. Even though relational schemas do not recognize networks set-owner-member relationships, cascading the key fields into the relations by the convention of key attributes captures this relationship. Upon completion of the cascading sequence, the number of attributes is set to equal to the number of attributes in the associated network database plus the cascaded fields. The mapping and cascading continues until all the relations are completed. Upon completion, control is returned to LIL where the user can query the database via SQL transactions.

V. MAPPING SQL STATEMENTS TO ABDL STATEMENTS FOR ACCESSING A NETWORK DATABASE

In Chapter IV we have discussed the schema transformation process needed to implement the mixed-process strategy. Figure 19 depicts the scheme used to complete the cross-model accessing of an AB(network) database. The forms in solid lines represent existing implementation and those in broken lines represent our work. The relational schema presented to the user is transparent in that the network schema appears to be a schema for a relational database. When a user creates an SQL transaction to access or update an AB(network) database, the existing Relational/SQL interface cannot be used, since it translates the SQL transaction into an ABDL equivalent for an AB(relational) database. Ours is an AB(network) database. Thus, the new network/SQL interface uses the original relational/SQL interface with some modifications which allow the interface to identify the schema created for the AB(network) database, instead of the ones for the AB(relational) databases. Note that the entire process does not involve the the network/CODASYL interface at all.

A. THE TRANSLATION PROCESS IN LI

Zawis [Ref. 18] presented two methods that the new language interface (LI) can be implemented. The first

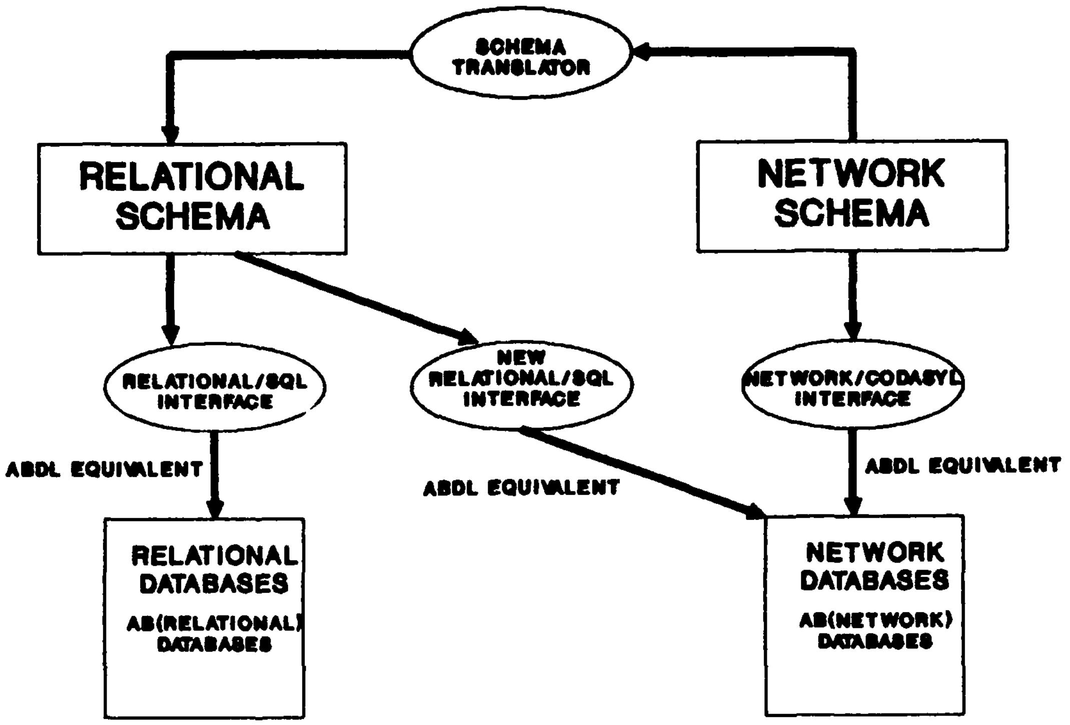


Figure 19. The Cross-Accessing Language Interface Design

method is to created a separate language interface (LIL,KMS,KC and KFS) for each data model. The LI used would be determined by the database model selected. The second method is to use the existing LI and modify it to by the user input. The second method is to use the existing LI and modify it to branch to the appropriate translation and processing activities as required by the user input. The latter method requires less implementation. Zawis chose to modify the existing LI to reduce the code size required. Similarly, we choose the same path. The consistency in improvements to MDLS and the existence of previous work are primary factors in determining the chosen method.

The SQL statement is sent to the relational/SQL interface. The relational/SQL interface branches to an internal network/SQL interface that ensures the needed checks and modifications of a network database are performed in KMS.

1. Query Processing in the LI

The user will load a database or select an existing database to process. The user is then given a menu to choose follow-up actions:

```
Enter mode of input desired
(f)--read in a group of transactions from a file
(t)--read in transactions from the terminal
(x)--return to the previous menu


ACTION ----> _
```

The user can use a prepared list of queries or create queries from the terminal. In any case a list of transactions are displayed on the terminal, with a number associated with each query. The following action menu is presented to the user:

```
Pick the number or letter of the action desired
(num)--execute one of the preceding queries
(d)--redisplay the list of queries
(x)--return to the previous menu


ACTION ----> _
```

The user can now select a query to process. The order of processing is not important if the query does not rely on another query. As an example a user cannot retrieve a file if it has not been created first. The query is sent to the kernel mapping system (KMS) for translation, and then to the kernel controller (KC) for execution. The results are returned to the user on the terminal and the action menu is presented for user selection.

2. Query Processing in LI

Previous theses provide descriptions of the query process in detail. In Benson and Wentz [Ref. 14], Emdi [Ref. 16] and Kloepping and Mack [Ref. 12] detailed descriptions of KMS are presented. The following is an overview of the query process based on the designers and implementors of MLDS. Zawis [Ref. 18] composed an excellent overview and we follow that overview.

SQL transactions are sent to KMS from LIL. The KMS function is to: (1) parse the SQL query and ensure the query syntax is correct, and (2) translate the query into an equivalent ABDL transaction. A valid query is sent to the kernel controller for processing by the kernal database system KDS, i.e., MBDS.

The KMS parser uses the Yet-Another_Compiler_Compiler (YACC). YACC is a program generator that performs a process on a stream of tokens that produces a parser that is syntactically correct. The Yacc-produced parser is a finite-state machine and performs a top-to-bottom parsing. The parser searches from left to right and with one token look-ahead. When tokens are recognized, portions of the output code may be executed or propagated up the hierarchy until a higher-level rule is satisfied. If the token string has successfully been processed then the parser terminates normally. If a syntax error is issued, the parser returns to the calling procedure.

The KMS data structure primarily consist of five data structures used during the parser process. The rel_kms_info is depicted in Figure 20. This structure contains information accumulated in the parser process to be used later. Attribute names used in the Select and Insert operations are held in the target list, the names of the relations being accessed are stored in templates, and Insert request attribute values are maintained in the insert list. The temp_str stores intermediate translation results and the join_str is needed to hold the translation of a second retrieve request of a join operation. The next_nest field is a pointer to the next rel_kms_info structure in the list of a nested Select transaction. The last field, alt_tgt holds information relating to the translation of non AB(relational) statements.

The four data structures pointed to by rel_kms_info are depicted in Figure 21. The relational KMS data

```
struct rel_kms_info
     {
     struct        target_list_info    *first_tgt;
     struct        templates_info      templates;
     struct        insert_list_info    *first_val;
     char                              *temp_str;
     char                              *join_str;
     struct        rel_kms_info        *next-nest;
     struct        alt_list_info       *alt_tgt;
     }
```

Figure 20. The rel_kms_info Data Structure

structures are covered extensively in Kloepping and Mack [Ref. 12]. Since we are accessing an AB (network) database the modifications are primarily concerned with the KMS parser. The branching in KMS is modified so that the SQL-ABDL AB (network) model. The remainder of the chapter describes the design considerations and implementation details involved in mapping the four primary SQL transactions into AB (network) equivalents.

B. THE SELECT STATEMENT

1. The Design

The SQL select command retrieves information from a database. Retrieval of information does not alter the database. Information is retrieved without modification,

```
target_list_info
    {
    char                              name(ANLength + 1);
    char                              tgt_rel(RNLength +1);
    struct    target_list_info        *next_attr;
    }
 templates_info
    {
    char                              name1(RNLength + 1);
    char                              name2(RNLength + 1);
    }
 insert_list_info
    {
    char                              *value;
    struct    insert_into_info        *next_val;
    }
 alt_list_info
    {
    char                              name(ANLength + 1);
    char                              op(RNLength + 1);
    struct    alt_list_info           *next_attr;
    }
```

Figure 21. KMS Parser Data Structures

when accessing an AB (network) database. The transformation of the AB (network) schema into an equivalent AB (relational) schema provides the user with a view of the attributes required to retrieve information, the key attributes are cascaded in the mapping. The DBKEY used in the CODASYL language interface is system generated and not provided to the user and therefore the DBKEY for the record must be added to the retrieve request before it is sent to the MBDS. The modification is not needed at root records, since theydo not have owner records and the relational user is given the actual attributes of the record.

2. An Implementation

Database integrity is not an issue in the select statement. The method used to implement the select is to branch to a retrieve_net procedure in KMS and add the record DBKEY to the ABDL request and send the request to MBDS. Figure 22 is an example of a select transaction and its equivalent ABDL transaction.

```
                SELECT sno,sname
                From sa
                WHERE  city = 'London'
    Retrieve((TEMP = SA) and (CITY= London)) (SNO,SNAME) BY
                DBKEY
```

Figure 22. A SQL Select Transaction

C. THE INSERT STATEMENT

The SQL insert statement is used to add information to an existing database. The database is modified and the AB (network) database integrity may be violated by the insert statement. The record-type-and-set-type relationships have to be maintained. The relational user does not know the notion of record types and set types. Nor does the relational user know their relationships and the restrictions that apply.

1. The Design

As mentioned in Chapter II the implementation of the AB (network) model is restricted to certain options. Here, the type of insertion is automatic in the AB (network) model. This means that insertion is based on set selection criterion. That is an occurrence of a record may not be inserted into a member record if a set type does not exist for the occurrence. The network/SQL language interface is needed to determine if the insert statement will violate the AB (network) integrity. The interface must retrieve the record DBKEY and add it to the insert statement before sending the insert statement to MBDS. The SQL user inserts the following transaction:

```
INSERT INTO sp (sno,pno,qty):
<'S1','P1',300>
```

The AB(network) model only allows for automatic insertion. If the insert is into a member record proper caution is taken to ensure that the record is placed in the proper set occurrence. The set selection mode must be considered.

a. The STORE-by-Application Statement

This method looks for proper set occurrence and then inserts the record.

b. The STORE-by-Value Statement

This method adds the requirement that the owner of the proper set occurrence must be located prior to insertion.

c. The STORE-by-Structure Statement

This method is similar to store-by-value except that the values of the owner and member attributes must match.

KMS only allows for the store-by-application method of set selection to be used in the present implementation.

2. An Implementation

By design, the AB(network) database accepts inserts if all set occurrences are proper. The insert_rel_to_net procedure traverses the AB(network) database to determine if the insert is valid. The first step is to evaluate whether the record type is a root record(owner only) or member record (can be owner also). This is accomplished by searching the set types using member name to match the record type. If no match occurs then the record type does not exist or it is a root record. In the latter case, the insert statement is then processed. Conversely, if a record type is found, it is a member. The insert must match the owner(s)' set type(s). This requires that a search of the owner(s) record(s) of the member record have the key attribute(s) used in the insert statement in the owner(s) record. This is accomplished by a RETRIEVE request for each owner record based on the attribute on which the set type is based. Each RETRIEVE is sent to a buffer to be sent to MBDS.

When the AB (network) translation is complete the parser completes its operations and control is given back to LIL. The KC then receives the linked list of ABDL requests from LIL. KC recognizes that an AB(network) database is being accessed and branches to a procedure that passes the RETRIEVE requests to MBDS. The record retrieved, if any, is sent to a buffer. The buffer is checked for at least one record. If the buffer is not empty the insert transaction is transmitted to the KC for processing. If the buffer is empty the user is informed that the insert as requested will violate the AB(network) database.
The following is a sample of the terminal display that the SQL user receives.

UNABLE TO COMPLY WITH REQUEST--to insert a network member record, an owner record must exist.

D. THE DELETE STATEMENT

1. The Design

The purpose of the SQL Delete is to delete records from a relational database. However, deleting records from a network database involves more than just deleting records. First, a database modification (in our case, deletion) requires checking to ensure that the network integrity is maintained. This process involves checking whether a target record is a parent record, if it is a parent, all the children must also be deleted. Second, because a network database has many-to-many relationships between parent-child records; it necessary to update (i.e., delete) all the associated occurrences in subsequent tree structures. Third, translate the SQL Delete into a number of AB(network) Deletes. For example, suppose a user performs the following Delete transaction on the network database in Figure 23:

```
DELETE SA
WHERE  SNAME = 'IBM'
```

If the supplier record 'IBM' is deleted from SA, the occurrence of 'SS2' in SP no longer has an associated parent in SA (i.e., integrity violation). In our design, we will delete the specified record and all associated occurrences.

The primary tasks in executing a SQL Delete transaction are to provide integrity checks on the network database, translate the SQL Delete to an equivalent set of AB(network) Deletes and finally perform the delete transaction.

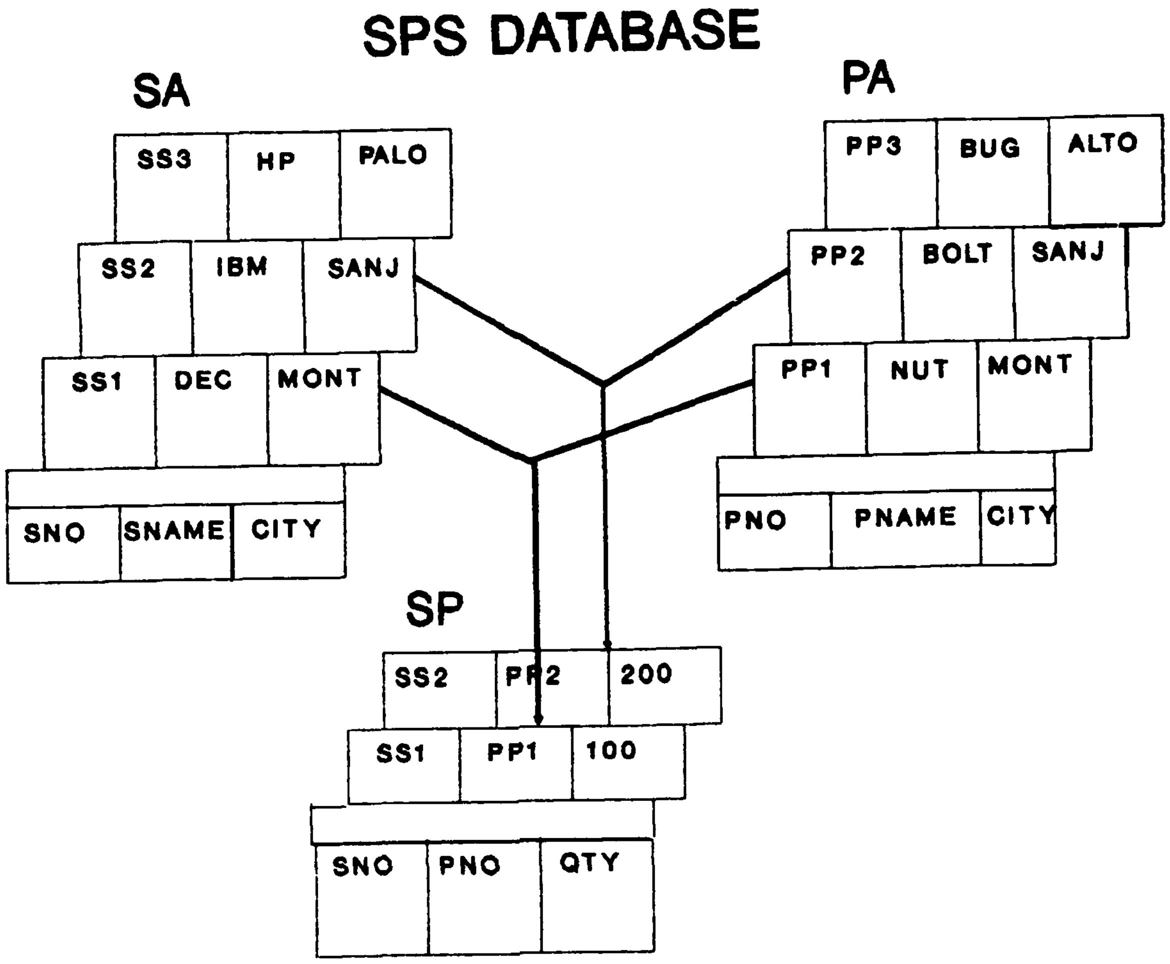


Figure 23. A Sample Network Database Prior to a Delete Operation

2. <u>An Implementation</u>

As described earlier, prior to executing a SQL (delete), a sequence of events must be accomplished. One of these events is integrity checking. Integrity checking consists of multiple deletes on parent-child related records. These multiple deletes are accomplished by buffering of intermediate results associated with records at different levels in the network tree structure.

The existing hierarchical language interface, contains the necessary logical concept to perform the Delete operation in our network database. This interface will allow us not to duplicate and integrate different code (i.e., mixed-processing strategy).

The operations necessary to accomplish a Delete transaction is dependent on the location of the occurrence in the network tree structure. If the occurrence is located at a leaf node in the network structure, then only a single delete transaction is needed. However, if the occurrence is located in a non-leaf node position in the network tree structure, multiple Retrieve and Delete transactions are needed.

The Delete operation requires multiple retrieves because the user does not provide all the necessary information required for the operation. As a relational user, the user does not know what records are associated with each other (in given many-to-many relationships) in the network structure. We will use the Retrieve operation to gather all the records associated with the target delete record. The retrieves take place at each level of the network structure; the results are then stored in the KC (Kernel Controller) buffer for later processing. An example of the transactions required for the Delete operation discussed earlier is illustrated in Figure 20. The first value retrieved is <SS2,IBM,SANJ>. The key value, 'SS2' is then used to delete occurrences in SP. The Delete operation is now complete. This is a very simple example of a network database in order to keep the details of the implication to a minimum. However, in a more complicated network database, execution of the Delete operation would continue to sibling and child records of SP. A Retrieve operation is needed at each level utilizing the 'SS2' value from the previous Retrieve. Following execution of these Delete operations in SA, the processing returns recursively to the associated record buffers with the specified 'SS2' value. Upon completion of the branch, execution then goes to the next branch for retrieval and Deletion using 'SS2' value until no branches in the tree remain for processing. The deletes are then sent to MBDS for execution.

The execution of the Delete operation starts in the KMS parser. The user specified Delete transaction is parsed and verified to be a legal Delete operation. A legal Delete operation is an operation that the parser recognizes based on existing grammar rules in the parser. Once the parser verifies that the operation is a Delete on a Network database, execution then branches to a routine that converts the AB(relational) Delete to an equivalent AB(network) transaction.

The remaining AB(network) Delete transaction now needs data structures in order to create and execute the Delete operation. These structures are built from the network database schema. The transaction now is complete and is sent to the KC for execution.

Upon completion of the Delete, the network data structures are released and the allocated memory returned to the operating system. The KMS then resumes processing and the relational data structures are re-initialized. In completion, control is returned to LIL for input from the user.

```
[RETRIEVE ((TEMP = SA) AND (SNAME = IBM)) (SNO) BY SNO]

[DELETE ((TEMP = SA) AND (SNO = **))]

[RETRIEVE ((TEMP = SP) AND (SNO = **)) (SNO) BY SNO]

[DELETE ((TEMP = SP) AND (SNO = **) AND (PNO =**))]
```

** is the place-holder for the value of SNO supplied by the prior retrieve statement.

Figure 24. A Sample AB(network) Delete Transaction

E. THE UPDATE STATEMENT

The SQL Update operation is used to modify attribute values in a relational database. If multiple values are to be updated, a sequence of Update transactions must be sent to MBDS. The Modify statement in Codasyl-DML is the equivalent to this (relational) statement.

1. The Design

The Update transaction is limited to non-key attributes. Non-key attributes are the attributes not needed to maintain the integrity of the network database. As a result, modification of key attributes values will cause corruption of the network database integrity. For example, if a key attribute field is changed in a record with children, the ancestor tree associated with the new value will be incomplete, and the existing children are no longer linked to a valid parent in the network structure. As documented in earlier work, our implementation of the Update operation remains consistent with this constraint.

2. An Implementation

The fact that we are constrained to only updating non-key attributes, our Update translation is achieved within the relational interface. LIL forwards the transaction to KMS for parsing and syntax verification. After the parser recognizes the transaction as a legal Update on a network database, a search routine is called to search the database schema for the desired attribute. If the target attribute is a key attribute, then the user will be sent the following message:

> UPDATE not allowed. Updates only allowed
> on NONE_KEY attributes only.

The request is aborted and control is returned to LIL for further user input. If the attribute found is a non-key field, then the Update is mapped to an equivalent AB(network) Update and passed to KC for execution.

In terms of future work, the Update could be modified to allow updates on key-attributes. This would consist of a series of Retrieve and Update operations similar to the Delete statement. Update statements could then be generated that will modify the cascaded key-attribute values of all descendent records in the network tree. In addition, it will be necessary to execute a retrieve on the ancestors of the record to be updated. The returned records are then stored in the KC buffer, thus, the complete ancestor tree for the new attribute value is established.

VI. CONCLUSIONS

Traditionally, the design and implementation of a conventional database system begins with the selection of a data model, followed by the specification of a model-based data language. An alternative to this traditional approach to database system development is the multi-lingual database system (MLDS). This alternative approach affords the user the ability to access and manage a large collection of databases via several data models and their corresponding data languages. This alternative approach has been designed and implemented at the Laboratory for Database Systems Research, Naval Postgraduate School, Monterey, California. Figure 25 depicts the multi-lingual database system.

MLDS restricts users to access individual databases with their respective data languages. For example, a network database can only be accessed via the network-data-model-based Codasyl-DML language. The extension of MLDS will support cross-model accessing of all the databases. The scope of this research is the design and implementation of an interface to support the access of a network database via SQL transactions.

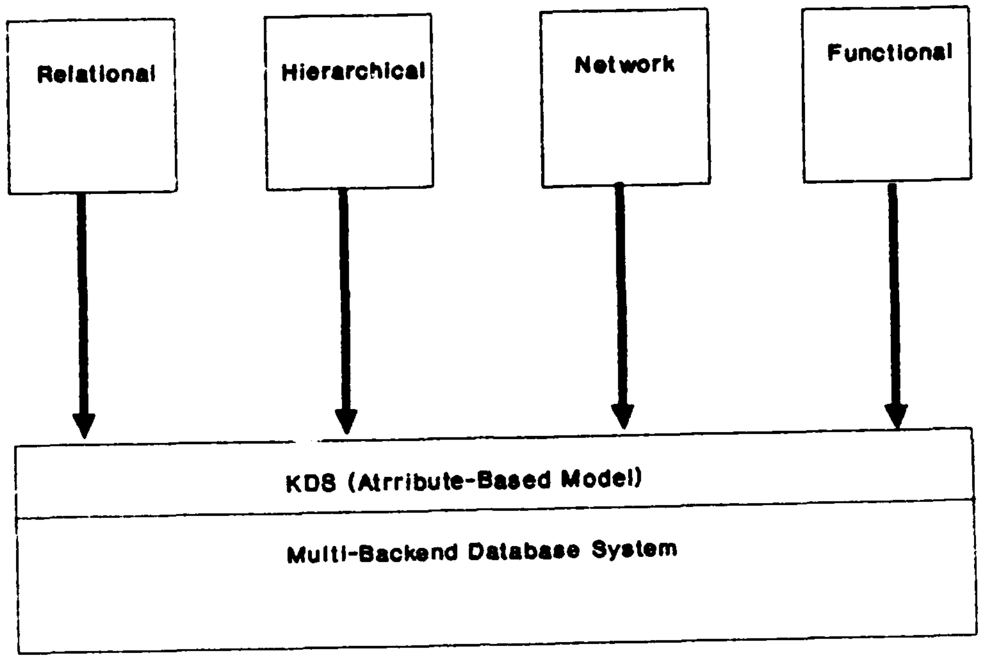


Figure 25. The Multi-Lingual Database System Concept

A. A REVIEW OF THE RESEARCH

We have presented three strategies for implementing this interface, inc luding high-level preprocessing, mixed-processing, and post-processing, prior to selecting the mixed-processing strategy as the most viable strategy. We have related our research with those research on cross-model accessing.

B. WHAT WE HAVE ACCOMPLISHED

The mixed-processing strategy involves two components. First, the schema transformation consisted of a methodology to map the network schema to a relational schema. This was accomplished by cascading key fields from the network schema to the relational schema; thus maintaining the owner-member relationships by keys. Second, we described the data structures and implementation details necessary to integrate the schema transformer in the Language Interface Layer (LIL).

The new language interface provides the capability of manipulating a network database via SQL transactions. This is accomplished by the translation of SQL transactions to an equivalent AB(network) transaction. We then detailed the changes to the existing relational-to-hierarchical language interface in order to provide us with cross-model accessing capability (i.e., relational-to-network). We then conclude our work by describing the four basic relational transactions, Select, Insert, Delete, and Update, in terms of the amount work entailed in the language interface and on the network database.

Our efforts at the Laboratory for Database Systems Research illustrates that a multi-model database system (MMDS) could be designed and implemented using existing software and the potential for further extension of MLDS is only limited by the motivation for research in this area.

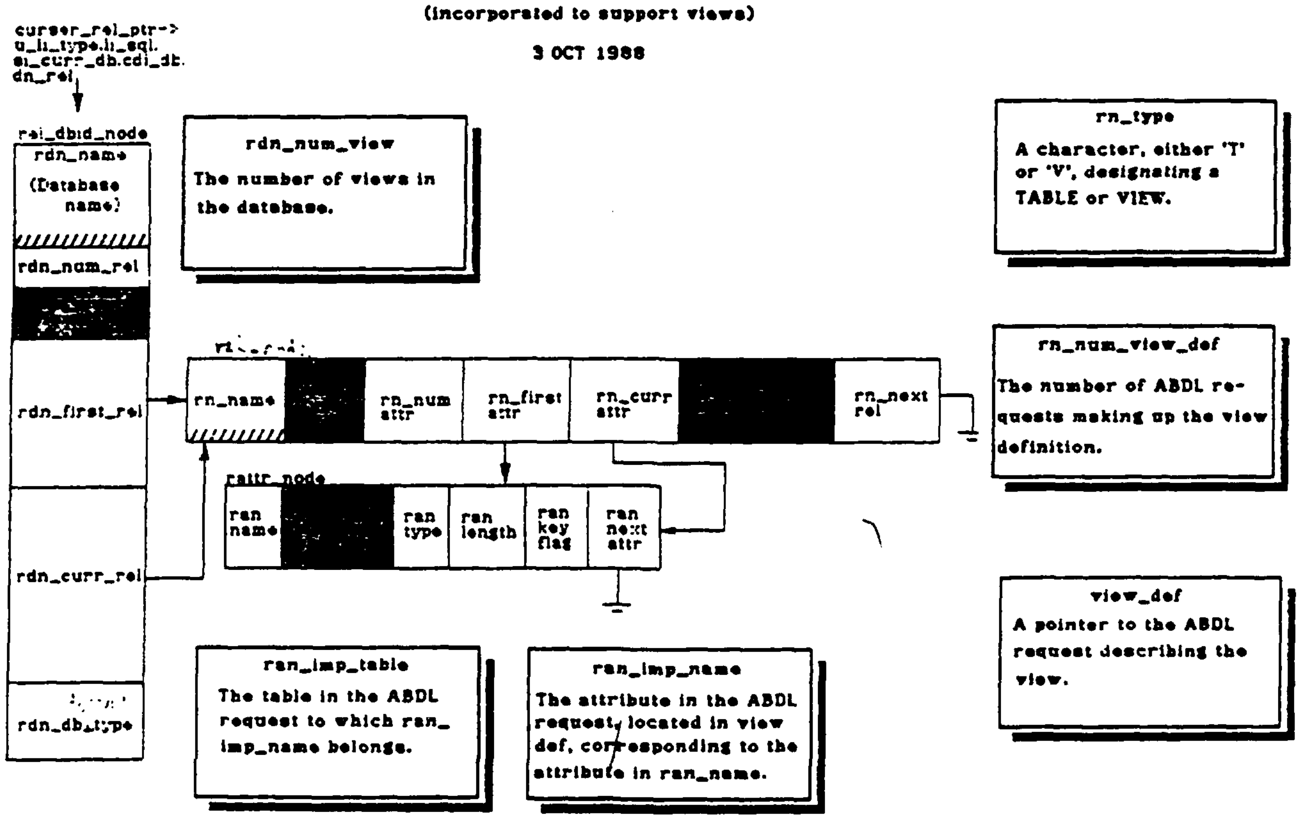
APPENDIX A

THE RELATIONAL DATABASE STRUCTURE

MODIFIED SYSTEMS CATALOG IN THE RELATIONAL/SQL LANGUAGE INTERFACE

(incorporated to support views)

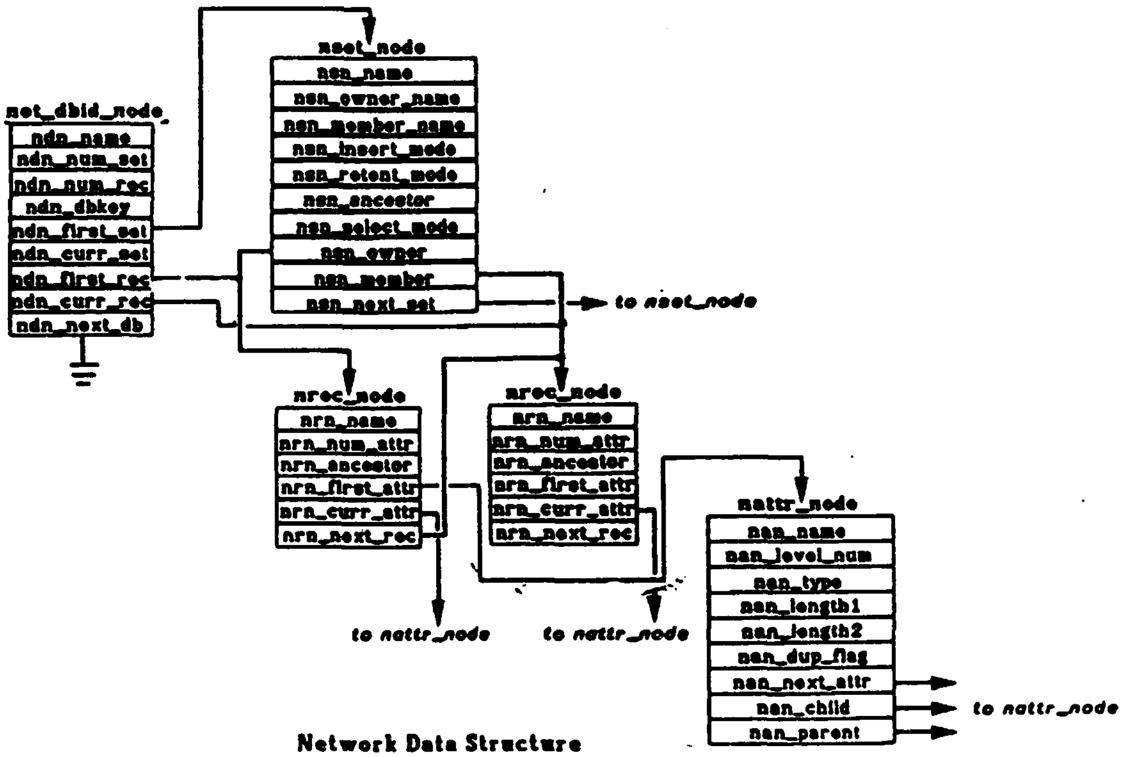
3 OCT 1988



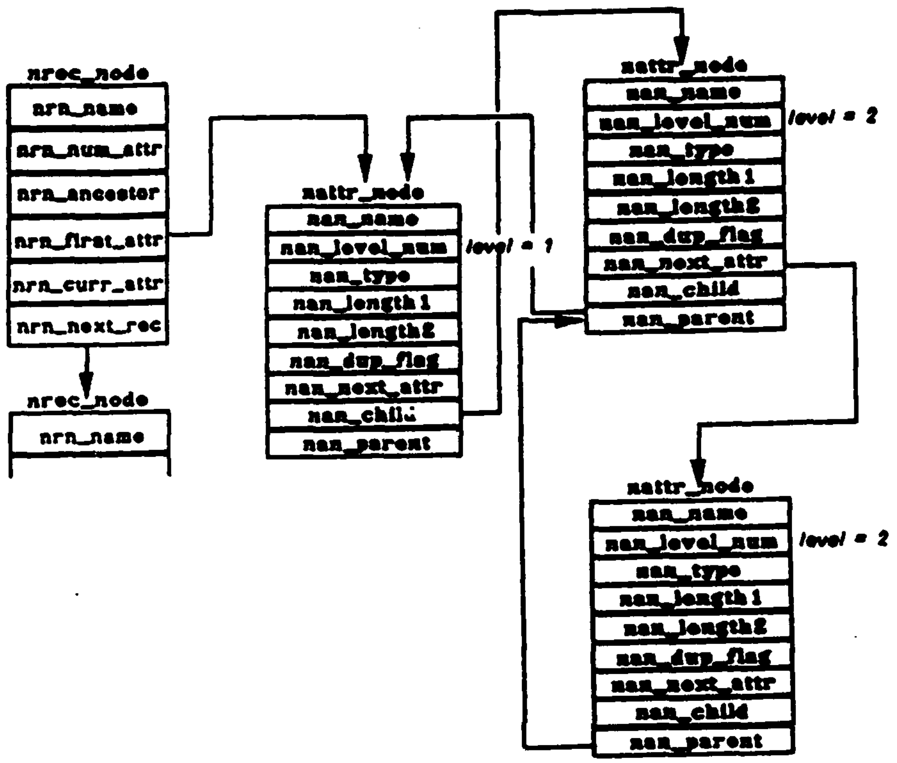
curser_rel_ptr->
u_li_type.li_sql.
si_curr_db.cdi_db.
dn_rel
rel_dbid_node
rdn_name
(Database name)
rdn_num_rel
rdn_first_rel
rdn_curr_rel
rdn_db_type
rdn_num_view
The number of views in the database.
rn_type
A character, either 'T' or 'V', designating a TABLE or VIEW.
rn_name
rn_num attr
rn_first attr
rn_curr attr
rn_next rel
rn_num_view_def
The number of ABDL requests making up the view definition.
rattr_node
ran name
ran type
ran length
ran key flag
ran next attr
view_def
A pointer to the ABDL request describing the view.
ran_imp_table
The table in the ABDL request to which ran_imp_name belongs.
ran_imp_name
The attribute in the ABDL request, located in view def, corresponding to the attribute in ran_name.

APPENDIX B

<u>THE NETWORK DATABASE STRUCTURE</u>



Network Data Structure



nattr_node structure

LIST OF REFERENCES

1. Demurjian, S.A. and Hsiao, D.K., "New Directions in Database-Systems Research and Development," Proceedings of a Conference on New Directions in Computing, IEEE Computer Society Press, August 1985.

2. Demuurjian, S.A. and Hsiao, D.K., "The Multi-lingual Database System," Proceedings of the Third International Conference on Data Engineering, IEEE Computer Society Press, February 1987.

3. Demuurjian, S.A. and Hsiao, D.K., "The Multi-Model Database System," Proceedings of the International Phoenix Conference on Computers and Communications, March 1989.

4. Hsiao, D.K. and Harary, F., "A Formal System for Information Retrieval from Files," Communications of the ACM, Vol. 13, No. 2, February 1970. Corrigenda, CACM 13,3 (March 1970).

5. Banerjee, J. and Hsiao, D.K., "The Use of a Database Machine for Supporting Relational Databases," Proceedings of the 5th Annual Workshop on Computer Architecture for Nonnumeric Processing, Syracuse, New York, August 1978.

6. Rollins, R., Design and Analysis of a Complete Relational Interface for a Multi-Backend Database System, Master's Thesis, Naval Postgraduate School, Monterey, California, June 1984.

7. Banerjee, J. and Hsiao, D.K., "The Use of a Database Machine for Supporting Relational Databases," Proceedings 5th Workshop on Computer Architecture for Nonnumeric Processing, August 1978.

8. Banerjee, J., Hsiao, D.K. and Ng, F., "Database Transformation, Query Translation and Performance Analysis of a Database Computer in Supporting Hierarchical Database Management," IEEE Transactions on Software Engineering, Vol. SE-6, No. 1, January 1980.

9. Banerjee. J. and Hsiao, D.K., "A Methodology for Supporting Existing CODASYL Databases with New Database Machines," Proceedings of National ACM Conference, 1978.

10. Rodeck, B.D., Accessing and Updating Functional Databases Using CODASYL-DML, Master's Thesis, Naval Postgraduate School, Monterey, California, June 1986.

11. Macy, G., Design and Analysis of an SQL Interface for a Multi-Backend Database System, Master's Thesis, Naval Postgraduate School, Monterey, California, March 1984.

12. Kloepping, G.R. and Mack, J.F., The Design and Implementation of a Relational Interface for the Multi-Lingual Database System, Master's Thesis, Naval Postgraduate School, Monterey, California, June 1985.

13. Weisher, D., Design and Analysis of a Complete Hierarchical Interface for a Multi-Backend Database System, Master's Thesis, Naval Postgraduate School, Monterey, California, June 1984.

14. Benson, T.P. and Wentz, G.L., The Design and Implementation of a Hierarchical Interface for the Multi-Lingual Database System, Master's Thesis, Naval Postgraduate School, Monterey, California, June 1985.

15. Wortherely, C.R., The Design and Analysis of a Network Interface for a Multi-Backend Database System, Master's Thesis, Naval Postgraduate School, Monterey, California, December 1985.

16. Emdi, B., The Implementation of a CODASYL-DML Interface for a Multi-Lingual Database System, Master's Thesis, Naval Postgraduate School, Monterey, California, December 1985.

17. Goisman, P.L., The Design and Analysis of a Complete Entity-Relationship Interface for the Multi-Backend Database System, Master's Thesis, Naval Postgraduate School, Monterey, California, December 1985.

18. Zawis, J.A., Accessing Hierarchical Databases Via SQL Transactions in a Multi-Model Database System, Master's Thesis, Naval Postgraduate School, Monterey, California, December 1987.

19. Hsiao, D.K. and Menon, M.J., "Design and Analysis of a Multi-Backend Database System for Performance Improvement, Functionality Expansion and Capacity Growth (Part I)," Technical Report, OSU-CISRC- TR-81-7, The Ohio State University, Columbus, Ohio, July 1981.

20. Hsiao, D.K. and Menon, M.J., "Design and Analysis of a Multi-Backend Database System for Performance Improvement, Functionality Expansion and Capacity Growth (Part II)," Technical Report, OSU-CISRC-TR-81-8, The Ohio State University, Columbus, Ohio, August 1981.

21. Codd, E.F., "A Relational Model of Data of Large Shared Data Banks," Communications, ACM, Vol. 13, No. 6, June 1970.

22. Date, C.J., in An Introduction to Database Systems, Addison-Wesley, 1981, 3rd edition.

23. Cardenas, A.F., in Data Base Management Systems, Allyn and Bacon, Inc, 1985, 2nd edition.

24. Olle, T.W., in The CODASYL Approach to Data Base Management, John Wiley & Sons, Ltd., 1978.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93943-5002	2
3.	Department Chairman, Code 52 Department of Computer Science Naval Postgraduate School Monterey, California 93943-5000	1
4.	Curriculum Officer, Code 37 Computer Technology Naval Postgraduate School Monterey, California 93943-5000	2
5.	Professor David K. Hsiao, Code 52Hq Department of Computer Science Naval Postgraduate School Monterey, California 93943-5000	2
6.	Thomas Chu 1140 Pebblewood Way San Mateo, California 94403	1
7.	Marciano Pitargue Vitalink Communications Corporation 6607 Kaiser Drive Fremont, California 94555	1
8.	Stanley and Jeanette Wade Route 1 Box 320 Martinsville, Virginia 24112	1
9.	Pamela Woods Route 6 Box 1796 Danville, Virginia 24541	3
10.	Richard W. Walpole 1967 Glenover Drive Memphis, Tennessee 38134	4